Autonomic Computing for Spacecraft Ground Systems

Zhenping Li, Cetin Savkli
Lockheed Martin Space Operations
7500 Greenway Center, Suite 200
Greenbelt, MD 20770
Zhenping.Li@lmco.com

Lori Jones
NASA/Goddard Space Flight Center
Bldg 32, Room N220C
Greenbelt, MD 20771
Lori.L.Jones@nasa.gov

Abstract

Autonomic computing for spacecraft ground systems increases the system reliability and reduces the cost of spacecraft operations and software maintenance. In this paper, we present an autonomic computing solution for spacecraft ground systems at NASA Goddard Space Flight Center (GSFC), which consists of an open standard for a message oriented architecture referred to as the GMSEC architecture (Goddard Mission Services Evolution Center), and an autonomic computing tool, the Criteria Action Table (CAT). This solution has been used in many upgraded ground systems for NASA's missions, and provides a framework for developing solutions with higher autonomic maturity.

1. Introduction

The concept of autonomic computing is the ability of computing systems to manage themselves based on high level objectives from management. It is inspired by the human autonomic system that maintains an optimal internal state through self regulation, while adapting to the changing environment. The vision [1] of autonomic computing is necessitated by the explosive growth in network applications and information services that are increasingly complex, dynamic, and heterogeneous, which have led to profound changes in almost every aspect of our lives. Using autonomic computing to manage technologies will be crucial in meeting the challenges of increasingly complex computing systems that may reach the limit of the human capability to manage and maintain in the near future. This is particularly critical when systems require a timely and decisive response to the demands of rapidly changing environments. There have been considerable efforts in both industry and the academic world to investigate autonomic computing concepts, architecture, as well as applications [2].

Spacecraft ground systems provide an important testing ground for the autonomic computing concept. A spacecraft ground system is complex: it involves many processes and subsystems working together, such as the flight dynamics subsystem, data processing subsystem, scheduling and planning subsystem, and command, control and communication subsystems. It is distributed: the subsystems and processes within a system are generally in different geographical locations and interact and communicate with each other through networks. It is heterogeneous: a ground system generally consists of mainframe or legacy systems for data processing and product generation and workstations for command, control, and communications on different platforms and operating systems. It also runs in real time, which has a high standard of requirements for reliability, availability, maintainability, as well as performance.

The next generation spacecraft will be empowered with new capabilities to generate new products for remote sensing, as well as imaging with much higher data rates and volume, such as the next generation of the geostationary operational environmental satellites [3]. The ground system and operations will become more complex and demanding, and will process spacecraft data at the daily scale of tera-bytes or even higher in the future. Autonomic computing for spacecraft ground systems will not only provide the long term solution to confront this increasing complexity, but will also bring short term benefits to current spacecraft operations as well. Specifically, it increases the system reliability and security, enables automation and autonomy at the system level, and thus reduces the costs for system maintenance and operations.

An autonomic computing system generally consists of managed elements and autonomic elements. The managed element is generally a functional unit, a hardware or software system that provides certain services. The autonomic element captures the signals from the managed elements on its health and operational status, analyzes the data based on the existing knowledge and high level objectives from management, and plans and carries out the appropriate actions for self-configuring, self-healing, self-

protection, and self-optimization. There are considerable scientific and engineering challenges to bring this concept into the reality. For spacecraft ground systems, autonomic computing requires an architectural solution to create an autonomic computing environment, and tools or middleware to provide autonomic computing services. The architectural solution for autonomic computing should provide an open standard for the interfaces and protocols for the interactions and communications among the components in a heterogeneous environment. It should also enable self awareness, which should make the detailed knowledge of its components, operational status, as well as other necessary information, available for the decision making process in the autonomic elements. The autonomic computing tool should be scalable, efficient, flexible, and extensible to provide core services at the system level. The focus of this paper is to present the ongoing efforts at Goddard Space Flight Center (GSFC) to define a reference architecture referred to as the GMSEC architecture [4] and to develop a GMSEC component, CAT, for providing autonomic computing services by Lockheed Martin Space Operations.

2. GMSEC Architecture

The GMSEC architecture is a solution for spacecraft ground systems that facilitates new and cost effective approaches for system development, integration, testing, and operations to meet the growing challenges in the current and future NASA missions.

The main concept of the GMSEC architecture is component based with a centralized message oriented middleware (MOM) shown in Figure 1. MOM provides the message services common to all system components, such as the security, message filtering and routing, and guaranteed delivery. The message services include the point-to-point and multicast services through the publish/subscribe and request/response schemes. The applications or components communicate with each other through a standard application programming interface (API) to MOM using messages. Each message includes a specific subject name that categorizes the message. Components publish messages by subject categories. The components receive messages by providing the subject names to the message middleware. The message delivery mechanism by MOM can be either synchronous or asynchronous.

The GMSEC architecture represents a natural extension from existing ground systems, in which the interfaces and communications among the subsystems and processes are implemented through TCP/IP socket connections that are mostly system dependent and proprietary. Using the middleware solution to provide the common services to all subsystems enables the component development to concentrate on its business logic. The divide and conquer strategy simplifies both component and middleware development. It also provides the flexibility to allow missions to choose components and middleware that meet their own specific requirements.

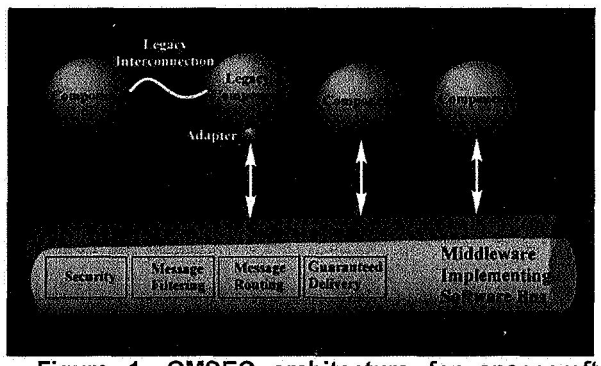


Figure 1. GMSEC architecture for spacecraft ground system

The GMSEC architecture standardizes the interfaces and protocols for the message deliveries through MOM, whose standard is open and non-proprietary. Experience in the Java enterprise computing standard, J2EE, that defines an open standard interface between the container and enterprise application component, shows that the open standard facilitates the technological innovations and infusions in the market place for both component and middleware development. This leads to the rapid development, deployment, and testing of enterprise applications at a much lower cost. The granularity of the coupling among components under the GMSEC architecture is higher than that in the standard component architecture, which leads to considerably simpler component integration and testing.

The GMSEC standardization efforts are two fold: the open standard API for the programming interface between the component and MOM that allows the point-to-point and multi-cast communications with certain levels of quality-of-service, and the standard schema for event message, telemetry, directive, data values, data transfer, and other types of messages. The GMSEC standard event message definition schema generally consists of a message header and a content section, which has gone beyond the traditional "time, type, fixed length text string" format, and provides much more content to allow new system monitoring capabilities. Key message definitions and reference implementations of the API in some commonly used

programming languages, such as Java, C++, and Perl, have been developed and released [4]. The reference implementation of the API converts proprietary interfaces of several MOM (middleware) tools on the market into the open standard interfaces on Windows, Linux, and UNIX operating systems.

To provide an autonomic computing environment at the system level, the GMSEC architecture has gone beyond the standardization of the interfaces and the message formats by establishing requirements for GMSEC compliant components: every component under the GMSEC architecture should be able to 1) publish event messages of its own operational status for real time monitoring and archiving, and 2) accept and process GMSEC standard directive messages. Components within the system may exercise discretion in what event messages they publish and what services they provide based on the number of attributes, including the source and authorization of the requestor. The expanded message definition, as well as the real time event log that covers every component in the system, enables system level monitoring and provides a very broad context to analyze the system performance. It also provides a very rich environment for data analysis and data mining to identify the correlations among the system components and system trends, and to anticipate the potential system problems. These requirements lead to a self-aware and interactive system that provides a standard for autonomic elements to interact with the managed elements, and enables the development of autonomic computing tools.

3. CAT Development under the GMSEC Architecture

CAT is a component under the GMSEC architecture with standard interfaces to MOM, and also a part of spacecraft ground systems. Thus, it should meet the general requirements for a component in both the GMSEC architecture and ground systems. These requirements are: the flexibility to manage any GMSEC compliant component, the scalability to monitor a system with many subsystems and processes, the extensibility to incorporate additional capabilities in the future, and the reliability and efficiency to perform in a real time environment. In addition, CAT should also be able to incorporate the knowledge accumulated in the existing spacecraft operations, which is particularly important for upgraded ground systems. This requires rigorous testing of autonomic computing tools. GMSEC has developed a laboratory for testing and simulating GMSEC compliant components, which primarily tests the robustness, reliability, and performance of a GMSEC component. The event analysis and monitoring tool, GMSEC Reusable Events Analysis Toolkit (GREAT) [5], has been developed for real time event monitoring, archiving, report generation, and event message generation for simulation and testing purposes. GREAT provides the necessary support to test and monitor the accuracy of the decision making process in an autonomic computing, real time environment.

3.1. The CAT Architecture

To meet these requirements, the system design and implementation of CAT are based on the best engineering practices and lessons learned in developing component and middleware solutions for both spacecraft ground systems and enterprise applications. CAT is implemented with Java and the latest J2EE technologies to ensure portability across operating systems, as well as rapid development from significant code re-use.

A layered approach for the CAT architecture is shown in Figure 2, which consists of three layers: the network layer, the service layer, and the configuration layer. The network layer captures all messages in MOM and forwards them to the service layer. At the same time, the network layer also accepts the actions generated by the autonomic agents in the service layer, and publishes them as GMSEC standard messages to MOM. The message could be a directive message to a specific component to change its behavior, or simply an event log message for monitoring, archiving and debugging purposes.

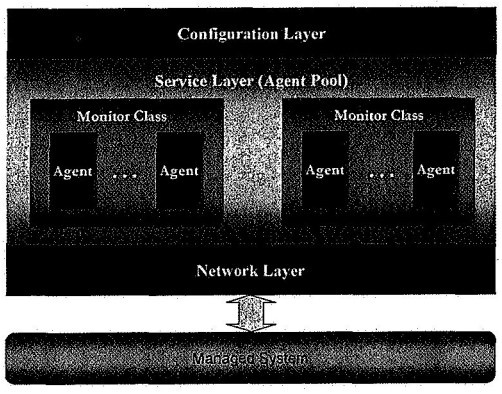


Figure 2 CAT architecture

The configuration layer is an XML file that can be configured during deployment or integration. The configuration file contains the domain specific information, rules and policies, as well as the knowledge base for a managed element. It also includes the necessary network information for the

network layer to interface with the message middleware. CAT also contains a tool that provides the ability to create, modify, and manage this configuration file through a GUI. The configuration file provides the inputs for the autonomic agents in the service layer that controls life cycles, internal states, and the decision making processes of autonomic agents, as well as determines the number of the autonomic elements at run-time. The configuration setup approach for the domain specific layer allows the operations personnel and management to setup the decision making rules based on their accumulated knowledge in spacecraft operations, which is important for upgrading the existing spacecraft ground systems. The schema for the CAT configuration will be discussed in detail in the next section.

The service layer is a component container, referred to as the agent pool. The components within an agent pool are monitor classes. A monitor class manages a service provided by components or entities within a system, and contains a group of autonomic agents that have the same lifecycles, rules for data analyses and decision making, and actions associated with decisions. Each agent within a monitor class manages a service provided by a single component or entity, has its own internal state, and runs as an independent thread.

The monitor class manages the lifecycles of its agents and provides the filtering capability to route the relevant agent in the monitor class. The agent pool provides mechanisms for fine grained collaboration among the agents within the same agent pool.

Life cycle management is very important in maintaining the efficiency of CAT and ensuring its scalability. An autonomic agent is created dynamically by an incoming message that meets certain criteria, and it can be terminated if the internal states of an agent satisfy a set of rules. Once an agent is terminated, it is removed from the agent pool by a pre-defined action. The lifetime for some agents could be very short, such as the agents that monitor the limit violations of spacecraft mnemonics, while the agent for monitoring the health and safety of a component in a ground system will remain active as long as the corresponding component remains active.

Message filtering and routing ensure that the autonomic agents only process the relevant incoming messages from their managed elements. This is particularly important since the message traffic in the middleware can be heavy in real time, and most of the messages in the traffic are not relevant for a particular agent in the agent pool.

CAT provides the mechanisms for both fine and coarse grained collaborations among the agents. The fine grained collaboration enables direct access of the internal states of one agent by the other agent within the same agent pool, while the coarse grained collaboration among agents in the same agent pool or different agent pools is achieved by exchanging the information through the event message publishing and monitoring scheme. For example, one agent could publish its own internal states to the message middleware as the event log message once its internal states have been updated, while the other agent could set up the configuration to monitor these states, and extract the data accordingly. The agent collaborations are very important at the system level monitoring to identify the correlations among the different subsystems, which provide comprehensive information on the system health and performance. For example, the power level of a spacecraft depends on whether it is facing the sun or in the dark, as spacecraft generally use solar power. The collaboration between the agent that monitors the power level on the spacecraft instruments and the agent that monitors the positions of the spacecraft in the flight dynamics subsystem will provide complete contextual information on the spacecraft power status.

3.2. Data Processing within an Autonomic Agent

The data processing and decision making processes in an autonomic element generally have the local and global control loops [2] based on Ashby's Ultra-stable system. The local loop handles known environmental states based on the knowledge embedded in the elements, which maps the environmental states to its behaviors. When an environmental state changes, the autonomic element will automatically generate actions based on the existing knowledge and policies. The global loop can handle the unknown environment states. It generally involves machine learning, artificial intelligence and/or human intervention, which in turn generates the necessary knowledge base for the local loop. The same architecture has been used in the Learning Classifier Systems proposed by Holland [6]. One could create agents specifically dedicated to both local and global loops in CAT. The agent collaboration allows local agents to access the internal states of the global agents to modify the existing rules and policies.

The basis of the data processing and decision making in CAT is a standard representation, on which the data analyses and decision making can be performed. Generally, a set of attributes is used to represent the internal states of an autonomic agent, which can have integer, float, Boolean, and string types. The attributes can also have the customized time type, which are used regularly in a real time environment. The attributes for a given agent are

classified into two groups: the original attributes $\{\alpha_i^o\}$ and derived attributes $\{\alpha_j^d\}^k$. The original attributes are extracted directly from the incoming messages using the pattern matching technology. The values of derived attributes $\{\alpha_j^d\}^k$ are updated by

$$\{\alpha_j^d\}^{k+1} = f\left(\{\alpha_i^o\}, \{\alpha_j^d\}^k\right)$$

where the integer k represents the kth iteration of the update triggered by the incoming messages with specified patterns. The function $f\left(\{\alpha_i^o\}, \{\alpha_j^d\}^k\right)$ could be a simple mathematical expression, such as the trigonometry functions or exponential functions, or it could also be a routine for machine learning algorithms, such as the decision tree algorithm. This depends on whether the routine or function is in the CAT data processing library. Currently, a mathematical library containing some basic mathematical functions is included in CAT. This framework could be easily extended to include libraries containing the advanced machine learning algorithms, adaptive algorithms, or an inference engine.

Both derived attributes $\{\alpha_j^d\}^k$ and original attributes $\{\alpha_i^o\}$ represent the actionable data, on which an informed decision could be made. The decisions made in an autonomic agent are based on rules having both original attributes and derived attributes, and each rule is associated with several actions. There could be several rules for a given agent that corresponds to different internal states, which may require different responses or actions. The rule based autonomic agents are widely used for monitoring and steering scientific applications [7]. CAT provides the capability to perform additional data processing and analysis so that the data would be actionable, and the informed decision can be made based on the management rules and policies.

Figure 3 shows the data processing and decision making process in CAT. It starts with the extraction of the data from the fields of the incoming messages using the pattern matching technology to generate the original attributes. The incoming messages with specified patterns may also trigger the update of the values of derived attributes through the user defined rules, the mathematical manipulation, or other data analysis routines. The combination of the original and derived attributes forms the actionable data. The decision making combines the actionable data with the management policies or rules, which leads to the actions sent to the network layer.

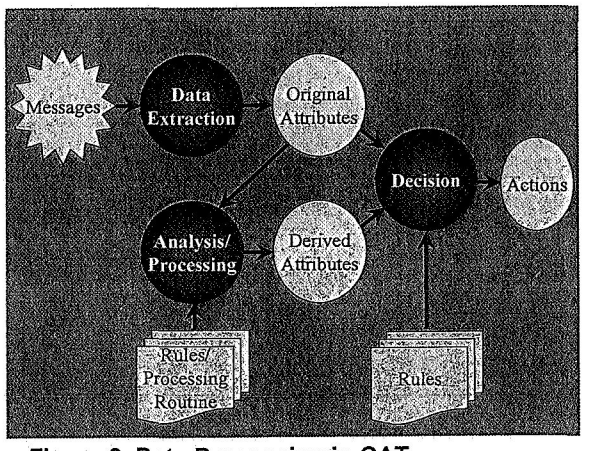


Figure 3. Data Processing in CAT

3.3. The Configuration Schema

The configuration file defines data processing, decision making, and the lifecycle of a particular agent. The basic unit for a CAT configuration is the monitor class, which defines a group of autonomic agents that manage the same service provided by different components. There could be as many monitor classes for a configuration as needed. A monitor class contains the following main sub-elements:

- **The constraint element** provides the filtering mechanism for an agent pool to process only the messages relevant to the attributes defined in the monitor class, and also ensures the messages to be processed come from the managed elements. This element is an optional feature to improve the processing efficiency.
- **The attribute element** defines both original attributes and the derived attributes. The element defines how the values of original attributes are obtained from the incoming messages.
- **The monitor trigger** element defines the rules for the agent pool to create an autonomic agent. It represents a logical relationship between the attribute value being extracted from the incoming message and the critical value defined by the user.
- **The primary key** is used to uniquely identify an autonomic agent within the agent pool, and it is created by combining the values of the original attributes in an agent. There is a one-to-many relationship between monitor classes and autonomic agents, since there may be several components that provide the same services.
- **The action element** provides the information necessary for autonomic agents to send either directive messages or log messages to the specified destination through the message

middleware, which is defined by the GMSEC standard.

- **The rule element** defines a set of conditions for both original and derived attributes, and action names that link to the action definition. The conditions are defined as the logical expressions for the relationship between the attributes extracted and the critical values defined by the user.
- **The function element** defines how a derived attribute is updated from the existing attributes through a combination of mathematical expressions, or an existing algorithm and rules.

In practice, not all elements listed here are needed for a given monitor class. If there is a one-to-one correspondence between the monitor class and an autonomic agent, the primary key entry is not needed. The monitor trigger element is not needed if the message has only one pattern monitored by the monitor class. For a simple monitor that requires no data processing, the equation element is also not needed.

For example, the configuration for a monitor class that monitors the heartbeat messages from components is shown in Figure 4. The availability of the mission critical component for continuous operations on a 24/7 basis is one of the crucial requirements for spacecraft ground systems. The subject and class constraints provide the filtering mechanism, which identify the messages with the specified patterns in their subject field and specified subfields to be processed by the agents. The two sub-elements within the same class constraint element have an "AND" relationship: if both patterns appear in their specified subfields of the incoming message at the same time, the requirements for processing the message are satisfied. The schema allows more than one class constraint element. The class constraint elements in a monitor class have an "OR" relationship. The primary key for corresponding autonomic agents is the component name that appears in the "COMPONENT" subfield of heartbeat messages. When an agent pool receives a heartbeat message from a new component, it automatically creates a new agent with the new primary key to monitor its heartbeat message. The required sub-element in the rule elements represents a logical expression; if the time since receiving the last heartbeat message is larger than 5 seconds, the action with the name GIVE_UP will be executed. The time variable *t_sinceReceivingLastMsg* is an internal attribute, which automatically resets when a new heartbeat message from the same component is received. The GIVE_UP action in the action element identifies the type of message as a GMSEC event log message, the destination of the message, and the entries in the specified message fields. The expressions ${attribute_name} will be replaced with the values of the attributes in the agent when the GMSEC log message is generated. The schema allows more than one action to be specified in a given rule. In practice,

```
<monitor-class name="HeartBeatMonitor" enabled = "true">
<subject-constraint>
<requirement attribute="SUBJECT" operator="~" value=".*C2CX.*"/>
</subject-constraint>
<class-constraint>
<requirement attribute="MESSAGE-SUBTYPE" operator="~" value=".*C2CX.*"/>
<requirement attribute="COMPONENT" operator="!~" value="CAT"/>
</class-constraint>
<primary-key>
<key order="0">component</key>
</primary-key>
<attributes>
<attribute name="component" type="String" field="COMPONENT" pattern="(.*)"/>
</attributes>
<rule name="GIVE_UP" enabled="true">
<act>GIVE_UP</act>
<requirement attribute="t_sinceReceivingLastMsg" operator=">" value="5"/>
</rule>
<action name="GIVE_UP">
<destination type="LOG">GMSEC.DEMO.LOG.CAT</destination>
<text field="SEVERITY">4</text>
<text field="MSG-TEXT">frequency=${t_sinceReceivingLastMsg} component=${component} Heart beat missing </text>
<text field="COMPONENT">CAT</text>
</action>
</monitor-class>
```

Figure 4. The Configuration for a Heartbeat Monitor

the actions include the directive to be sent to a backup component for the failover procedure, the log message, and an exit action that terminates the agent and removes it from the agent pool.

The heartbeat monitor class listed here is very simple and generic, but at the same time, very powerful. The agent pool manages the heartbeat autonomic agents for the whole system and is adaptive to the changing environment: it automatically creates an agent when the heartbeat message from a new component is detected, takes the failover action and then removes the agent from the agent pool in case of a component failure. As the failed component is generally off-line, the corresponding agent is no longer needed.

4. Autonomic Computing in Spacecraft Operations

Both the GMSEC architecture and the autonomic tool, CAT, have been deployed in many NASA missions in order to increase automation and autonomy, as well as reduce operational costs. The GMSEC architecture and the CAT tool have become a standard for ground systems in current and future NASA missions.

The autonomic computing solution for ground systems is used to replace operations personnel for monitoring and steering spacecraft operations. The self-configuring and self-healing capabilities of autonomic elements are crucial for fully autonomous or "lights out" operations. In the upgraded ground system for the Tropical Rainfall Measuring Mission (TRMM) spacecraft, CAT is used to monitor the health and safety data from the spacecraft. Flight operations personnel are informed if an error is detected, which may indicate a failure of either hardware or software on the spacecraft. Generally, there are hundreds or even thousands of parameters and attributes referred to as mnemonics that describe the health and safety of each hardware/software item on a spacecraft. Creating one agent for each mnemonic is simply not practical and inefficient; the combination of agents and a generic monitor class has reduced 180 rules to around 40 rules in CAT, and enables much more efficient processing in real-time. CAT is also used to monitor the heartbeats from mission critical components and to initiate a failover operation in case of a component failure.

As users get more familiar with CAT and its capabilities, more sophisticated scenarios for increasing the automation in their operations are being implemented. As part of the ground system automation effort for the Earth Observing System (EOS) satellite Terra, CAT performs the decision making to configure the ground system components for data acquisition and commanding before, during, and after the contact between the satellite and the ground stations. In particular, CAT will be performing the tasks normally performed by operators during the execution of procedures. Currently, the Terra procedures that are executed to configure the ground equipment for spacecraft contacts require operator inputs at various decision points during execution. These decision points will be monitored and executed by CAT in the new ground system. The same services for self-healing in the TRMM ground system will also be provided in Terra.

The actionable data obtained through data analysis in an autonomic agent provides the basis for decision making not only for the autonomic agents, but also for management as well. One could configure an agent that uses the data analysis capability to monitor system wide events for statistical collections and other useful data, and these data can be archived by defining an action to send a directive message to the archive component in the system. This is called *business intelligence* in enterprise applications. The summary report for spacecraft and ground activities can be generated automatically for management.

The architectural solution and autonomic computing concept have also been used in the ground system for the Small Explorer (SMEX) missions, which controls a constellation of small scientific spacecraft. Additionally, the upgrade of the ground systems for the other EOS satellites, the Aqua and Aura missions, is planned in the near future. The infusion of the GMSEC architecture and autonomic computing in other new mission ground systems is also planned.

5. Summary: Increasing Autonomic Maturity

The architectural blueprint for autonomic computing by IBM proposed an autonomic computing maturity model in 5 levels [8]: 1) basic, 2) managed, 3) predictive, 4) adaptive, and 5) autonomic. The capabilities provided by CAT under the GMSEC architecture suggest that the autonomic maturity for the current solution is between the predictive and adaptive levels. Increasing the autonomic maturity requires improvements in both the GMSEC architecture and CAT. The current GMSEC architecture does not go far enough in the standardization process to enable autonomic computing with a higher maturity.

To increase the autonomic maturity at the architectural level, the GMSEC architecture should be upgraded to the service-oriented GMSEC architecture

(SOGA). The component re-use paradigm in the current GMSEC architecture will be replaced by the service re-use paradigm. A service received from one component is obtained through a "locate, negotiate, and lease" procedure, which is also called a "find bind and execute" scheme. Thus, the service re-use enables completely plug and play components.

The open standard for the message delivery through the middleware under the GMSEC architecture is a very important step toward achieving SOGA. To upgrade the GMSEC architecture into a SOGA, a new standard ontology and protocol are needed for services, as well as quality of service, service discovery, and a service contract in the GMSEC standard messages. In addition, a service registry based on these standards needs to be developed as part of SOGA.

To ensure system awareness and an interactive environment for autonomic computing, the common attributes that represent the run-time properties of a service need to be defined and standardized. Thus, SOGA should require that a compliant component for a given service publish these attributes as the standard event messages when the values of these attributes change, and process directives that can change these run-time properties. Both messages for publishing these attributes and directive messages for changing component attributes should be standardized as well.

The monitor classes defined in CAT are autonomic elements that manage services. The same service in SOGA can be provided by several components with different qualities of service. Considering the heartbeat monitor class example, publishing the heartbeat message by each component in a system could be regarded as a universal service in a SOGA environment. Thus, the monitor class manages the heartbeat service regardless of the specifics of a component, and adapts to the changing environment. Because the heartbeat service in the GMSEC architecture is a standard, the same configuration can be used in any GMSEC compliant system, which makes it more adaptive, generic, and portable. The standardized service in SOGA will standardize monitor classes as services, which allows them to be re-used from one mission to another without significant changes.

The standardized event and directive messages for attributes in a service make it possible to define system level attributes for its overall performance, which could be functions of the attributes of different services in a system. Therefore, an optimal performance boundary could be specified by management or an administrator as overall objectives.

The machine learning algorithm and optimization algorithm could be introduced on this platform to establish the relationship between the optimal performance boundary, that could generally be multi-objective, and the attributes of services. When a new service component is connected with the message middleware, the autonomic agent could be created automatically, and the service attributes configured based on the optimal boundary.

There are still considerable scientific and engineering challenges ahead for an autonomic computing system. The GMSEC architecture and the autonomic computing tool, CAT, presented here are an important and significant step toward an autonomic computing solution for spacecraft ground systems. This approach will provide some useful lessons in developing autonomic computing solutions for other enterprise application systems.

6. References

[1] Paul Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology"; http://www-1.ibm.com/autonomic., Oct. 2001. Jeffrey O Kephart and David M. Chess, "The Vision of Autonomic Computing", IEEE Computer 35 (1); 41, 2003.

[2] Manish Parashar and Salim Hariri, "Autonomic Computing: An Overview", *UPP 2004,* Mont Saint-Michel, France, Editors: J.-P. Banâtre et al. LNCS, Springer Verlag, Vol. 3566, pp. 247 – 259, 2005 and references therein.

[3] See http://www.osd.noaa.gov/ for detailed information.

[4] See GMSEC project, http://gmsec.gsfc.nasa.gov for further information.

[5] Zhenping Li, Cetin Savkli, and Dan Smith, "Increasing The Operational Value of Event Messages", Proceedings of 5th International Symposium On Reduce the Cost of Spacecraft Ground System and Operations, July 8-12, 2003, Pasadena, California.

[6] J. H. Holland, "Adaptation", Progress in Theoretical Biology, eds. R. Rosen and F.M. Shell, Plenum, 1976.

[7] Hua Liu and Manish Parashar, "Rule-based Monitoring and Steering of Distributed Scientific Applications", *International Journal of High Performance Computing and Networking*, issue 1, Inderscience, 2005.

[8] IBM Corporation. An Architectural Blueprint for Autonomic Computing, http://www-03.ibm.com/autonomic/library.shtml. April, 2003.

Autonomic Computing for Spacecraft Ground Systems

Zhenping Li, Cetin Savkli
Lockheed Martin Space Operations
and
Lori Jones
NASA/Goddard Space Flight Center

Space Mission Challenges for Information Technology 2006

Pasadena, CA
July 17-20, 2006

Agenda

- Autonomic Computing
- GMSEC Architecture
- Criteria Action Table
  - A component approach to Autonomic Computing
- Applications
- Summary

LOCKHEED MARTIN

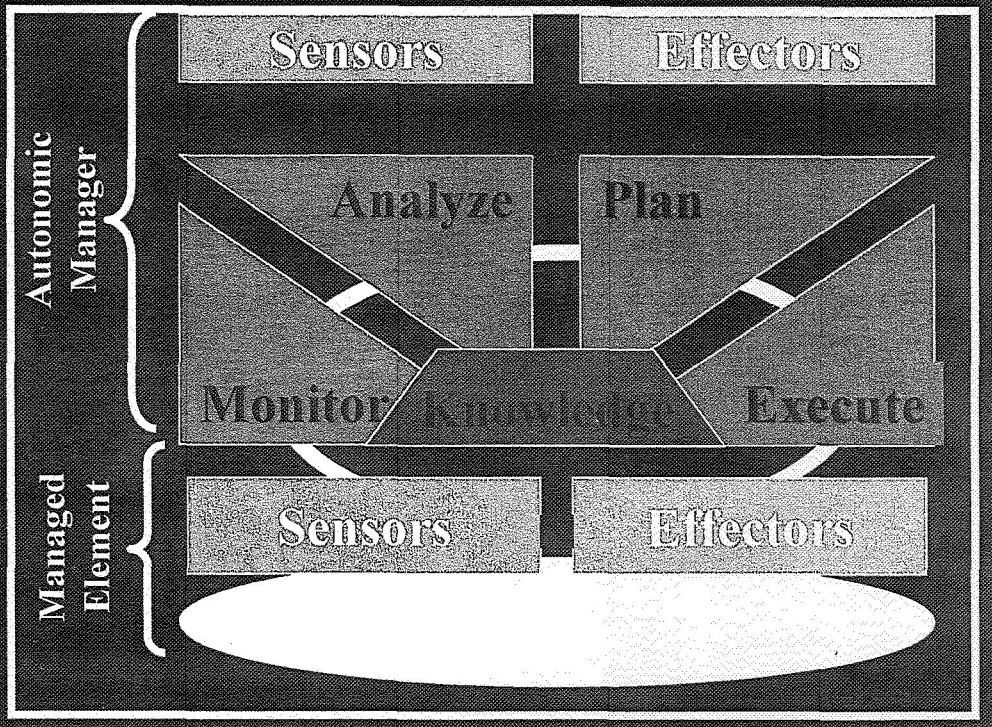
NASA

Autonomic Computing

- A self managing and regulating computing system that enables
  - Self configuring
  - Self healing
  - Self optimizing
  - Self protection
- A new computing paradigm to meet the challenges of complex computing applications that are increasingly complex, heterogeneous, and dynamic
- For spacecraft ground systems, autonomic computing provides a systematic approach for increasing system automation and autonomy
  - Reducing the operation and maintenance costs
  - Increasing system reliability
  - Provides a rapid response to a dynamically changing environment

Autonomic Computing Structure

- Autonomic elements
  - Provides autonomic computing services
- Managed elements
  - A component or service in the system
- Autonomic elements and managed elements form a feedback control loop
  - The managed element's behavior is based on management policies and rules



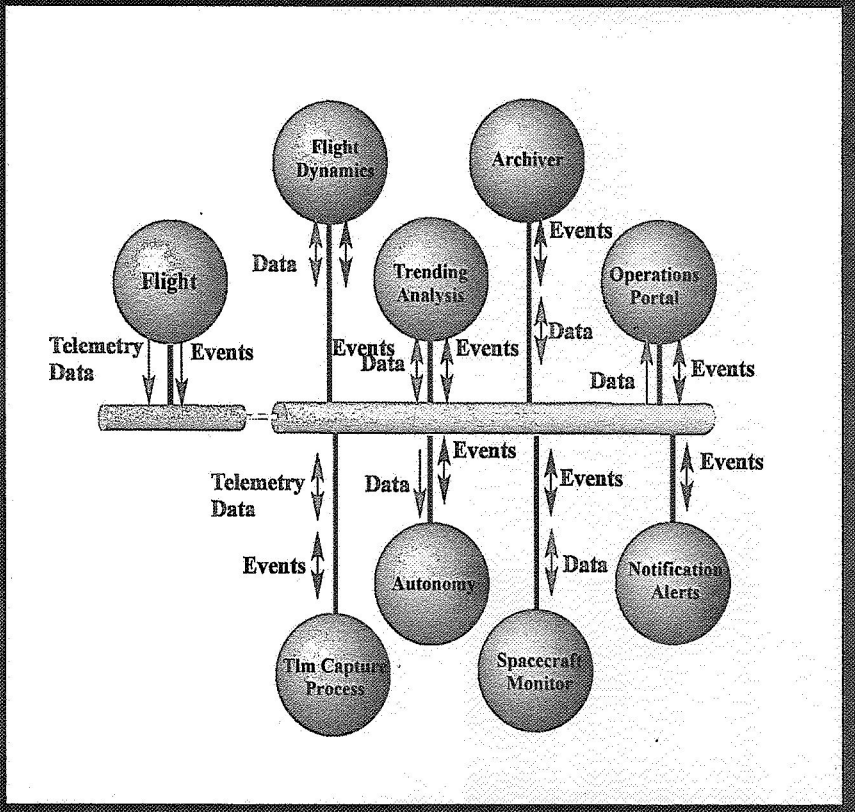
An Autonomic Element

Autonomic Computing for Spacecraft Ground Systems

- Components and services already exist in ground systems
  - How to develop the autonomic computing elements without modifying the existing components significantly?
- Require solutions at both the architecture and component levels
  - Architecture level:
    - Use open standard for the interaction between the autonomic elements and managed elements
    - Create system awareness
      - System event messages that contain all necessary monitoring information for the autonomic elements
  - Component level:
    - Provide autonomic computing services
    - Configurable
      - Provide flexibility to evolve and adapt, and to add additional management rules and policies

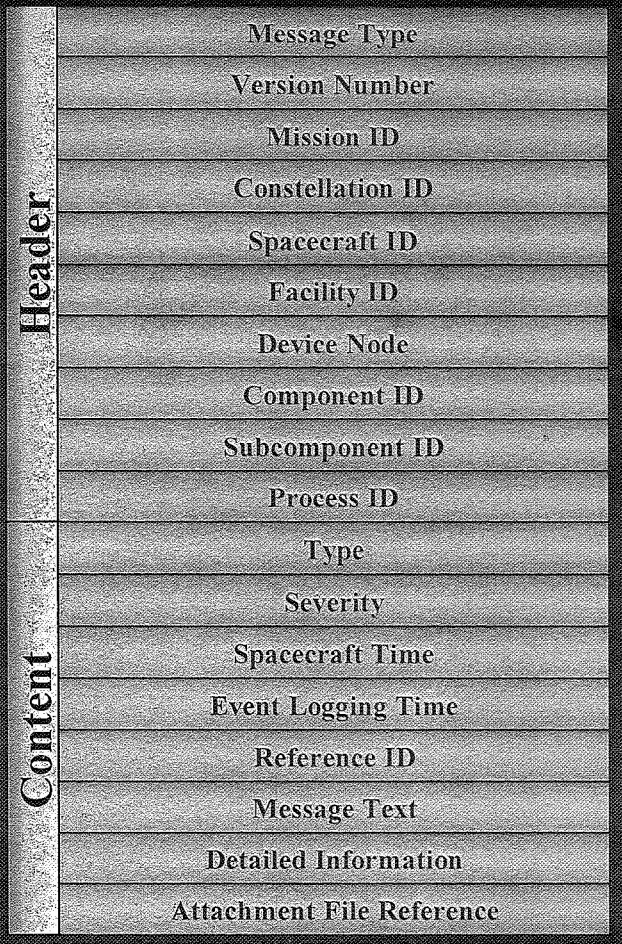
GMSEC Ground System Architecture

- Provides flexible and cost effective approach to meet the operational needs of current and future missions
- Middleware provides the services common to all components, and is a standard for the interaction and communication among the components
  - Message routing and delivery
  - Security and guaranteed delivery
  - Open standard for the interfaces between the component and middleware
    - Easy integration
  - Open standard for the messages
    - Standard communications among the components
    - Message Standard defined for the event message, directive request & response, heartbeat, and other types of messages
- Higher interoperability and portability for components
  - Missions select different components and middleware based on their requirements



System Awareness in GMSEC Architecture

- Message Standard goes beyond "time, type, fixed length String format"
  - More information available for data analysis and correlations
- Every component within the ground and flight system publishes its status
- Every component within the ground system should accept GMSEC directive messages to change its behavior
- Creates an autonomic computing environment to manage any component or service



LOCKHEED MARTIN

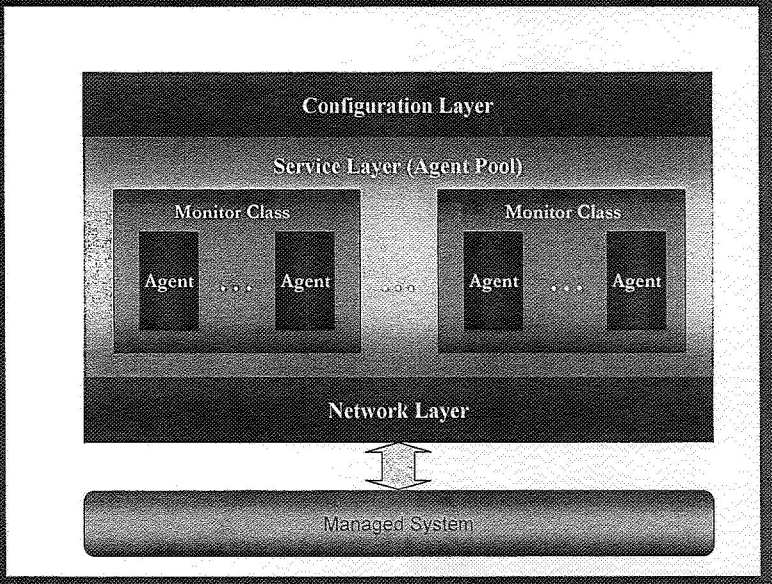
NASA

Autonomic Component: Criteria Action Table

- The managed elements are services provided by components within a system
- A component generally provides more than one service
  - Health and safety status service
  - Domain Specific services
  - The messages associated with a service and published by a component generally have a fixed template
    - Enables easy extraction of the relevant information from the messages
- Monitor class
  - A group of autonomic agents that manage the same service provided by different components or entities
  - Follows the same management policies and rules
  - Have the same life cycle
- Lifecycle of autonomic agents
  - Agents are created dynamically
    - Generally triggered by an event
    - Can be terminated if it is no-longer needed
  - More adaptive and flexible

CAT Architecture

- Configuration Layer
  - Xml configuration files that contain the domain specific instructions on how information is processed, the rules to apply to that information, and the responses that are required
- Service layer:
  - Agent pool manages the lifecycle of agents
  - Routes the relevant message to agents
  - Agents interact among each other within the agent pool
    - One agent can access the attribute values in the same or different monitoring classes
- Network Layer
  - Interface with GMSEC middleware
  - Monitor the system
  - Generate the directive or event messages that are sent to pre-defined destinations

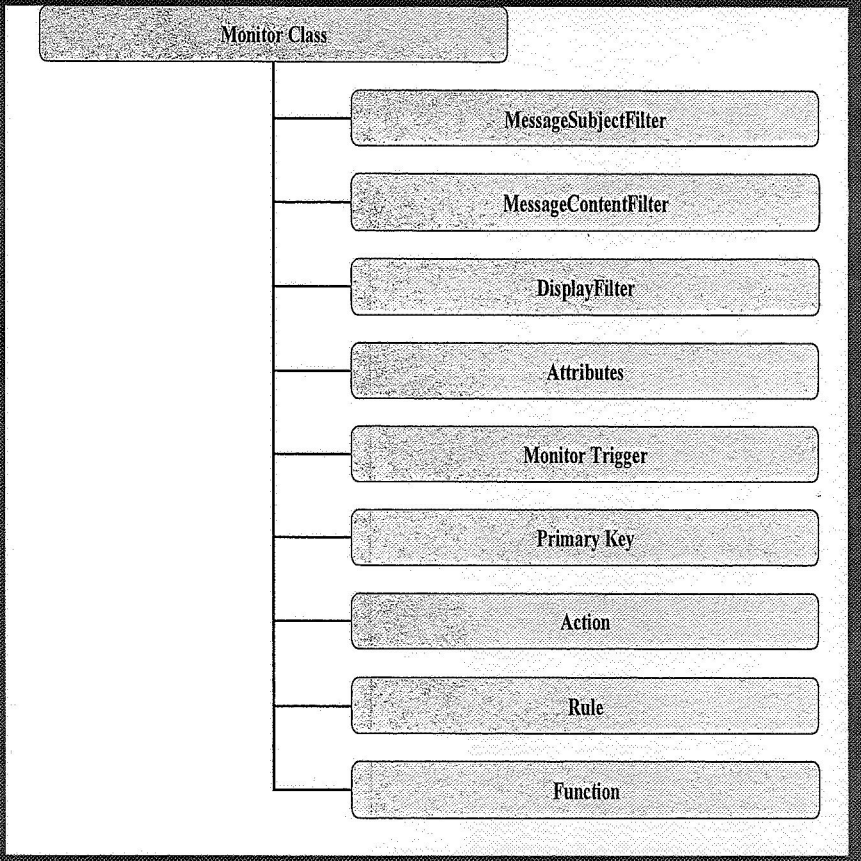


LOCKHEED MARTIN

NASA

The Agent Representation

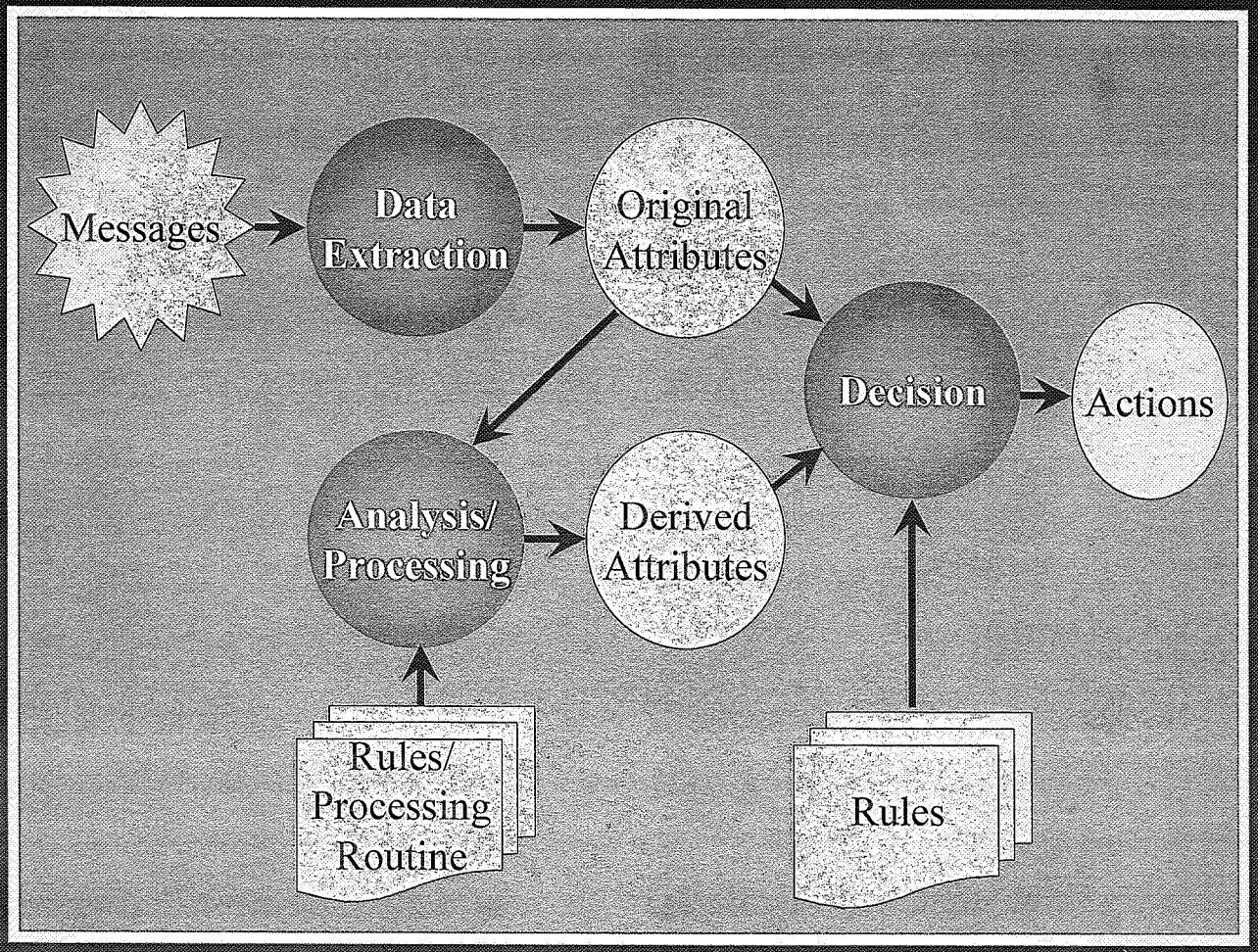
- Basic representation for data processing is the "attribute"
- Filters ensure only relevant messages are routed to the agents
- Monitor Trigger enables agents to be created dynamically
- Primary Key provides the unique identifier for an agent within a monitor class
- Function provides a platform for data analysis
  - Additional data analysis routines can be easily added
- Rules, using specific criteria, determine if pre-determined actions are to be taken
  - Actions, associated with the rules, provide the information on how the messages are to be constructed



LOCKHEED MARTIN

NASA

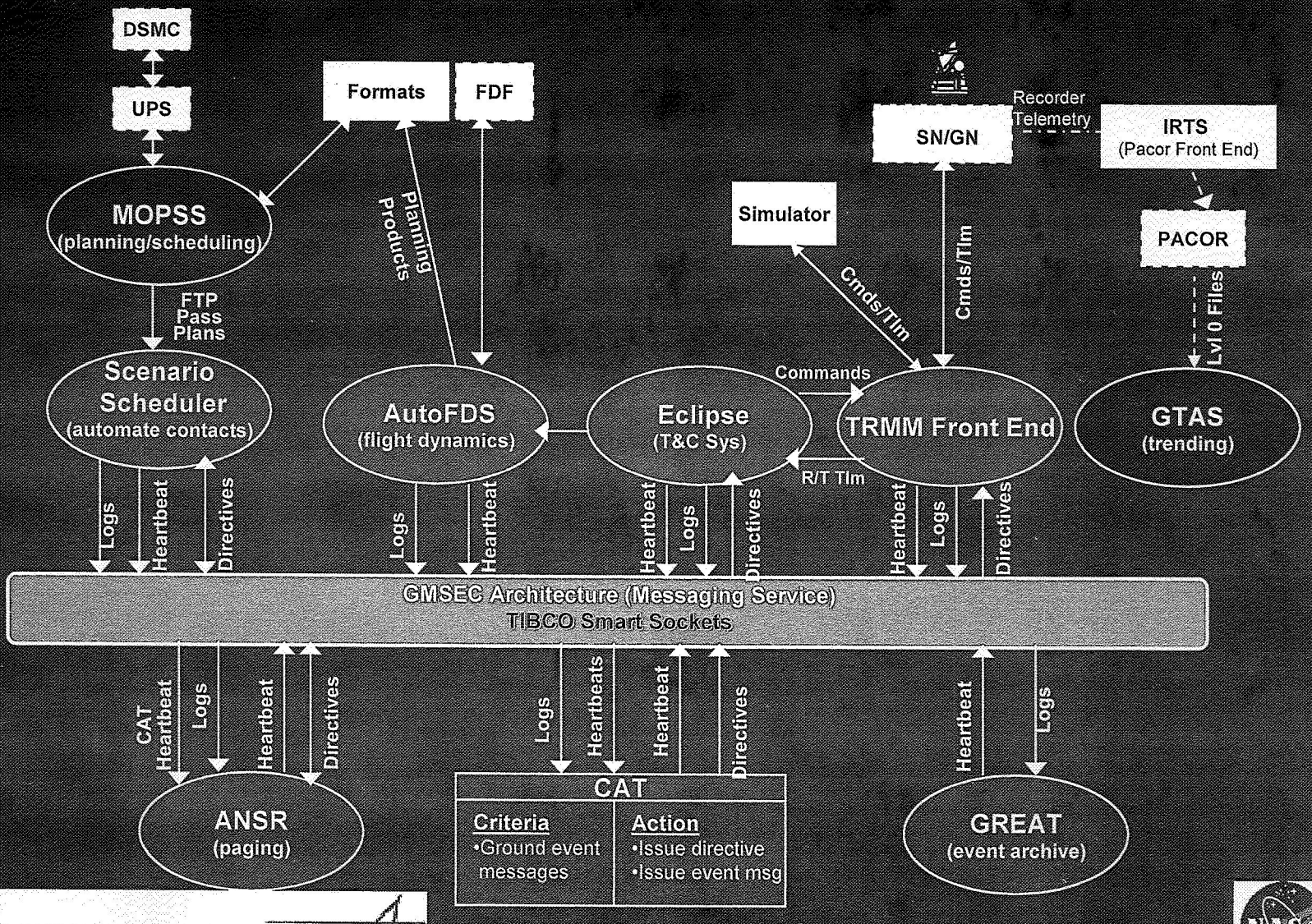
Data Processing



Current Applications

- GMSEC Architecture has become open standard for GSFC ground systems
  - Autonomic computing approach is the key for increasing automation and reducing the cost of operation and maintenance
- Existing Re-engineering efforts
  - Tropical Rainfall Measuring Mission (TRMM)
    - Reducing operational costs by 50 percent
    - Enabling lights out operations
  - Small Mission Explorer (SMEX)
  - Current EOS automation efforts on Terra, Aqua, and Aura
    - More complex configurations are being implemented
- New GSFC Missions
  - Discussion/Planning: LRO, GLAST
  - Sharing Tools and approach: JWST
  - Implementing: ST5
  - Co-developing: MMS and GPM
- Other NASA centers using GMSEC architecture approach as well

TRMM Reengineered Architecture



LOCKHEED MARTIN

NASA

Summary

- Autonomic Computing requires solutions at both architecture level and component level
- Increasing the autonomic maturity requires a service oriented architecture
  - Service monitoring becomes standard
    - More adaptive
  - System awareness is still required
- A new paradigm for increasing automation at the system level